ICASE

A MICROPROCESSOR ASSISTED GRAPHICS SYSTEM

Griffith Hamlin, Jr.

Thomas Crockett

Report Number 78-3

January 31, 1978

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING

NASA Langley Research Center, Hampton, Virginia

Operated by the

UNIVERSITIES SPACE USRA RESEARCH ASSOCIATION

A MICROPROCESSOR ASSISTED GRAPHICS SYSTEM

Griffith Hamlin, Jr.*

Thomas Crockett†

ABSTRACT

GRAF80 is a small scale graphics system designed to demonstrate some of the capabilities which are offered by microprocessor-assisted graphics. As such it employs both hardware and software techniques. The system implements in a microprocessor a subset of the features of the SIGGRAPH-ACM CORE graphics package, including picture segmentation and a subset of the output primitives and primitive attributes. Access to the write-through capabilities of the Tektronix 4014 terminal is also provided. The overall cost to add these features to a Tektronix 4014 is relatively small.

*Los Alamos Scientific Laboratory, Los Alamos, New Mexico 87545

†ICASE, NASA Langley Research Center, Hampton, VA 23665

This report was prepared as a result of work performed under NASA Contract No. NAS1-14101 while both authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

GRAF80 is a small scale graphics system designed to demonstrate some of the capabilities which are offered by microprocessor-assisted graphics. As such it employs both hardware and software techniques. The system implements in a microprocessor a subset of the features of the SIGGRAPH ACM CORE graphics package [1], including picture segmentation and a subset of the output primitives and primitive attributes. Access to the write-through capabilities of the Tektronix 4014 terminal is also provided. The overall cost to add these features to a Tektronix 4014 is relatively small.

HARDWARE

The hardware consists of an IMSAI 8080 microprocessor which forms the heart of the system. It acts as a dedicated device driver with software supporting picture refreshing and segmentation. Any one of several other available 8-bit microprocessors could be used as well. A Tektronix 4014 direct view storage tube terminal is used as the graphical output device. Several possibilities exist for interconnecting the microprocessor, Tektronix terminal, and host computer. Figure 1 shows one possibility which uses only commonly available equipment.

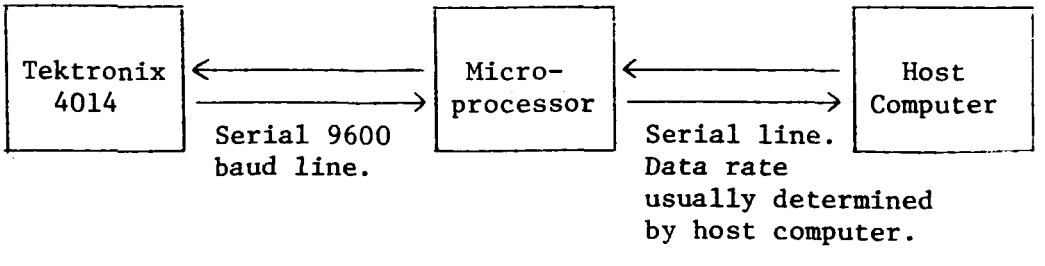


FIGURE 1

The Tektronix 4014 will support serial communications up to 9600 baud with the microprocessor. This rate should be sufficient to provide interactive access to images stored in the microprocessor's segmented display file. It is not sufficient to refresh images using the write-through mode of the 4014. To remedy this, we have constructed a simple parallel interface to the 4014 and connected it as shown in Figure 2. This provides byte parallel transmission at the Tektronix 4014 MINIBUS® speed.

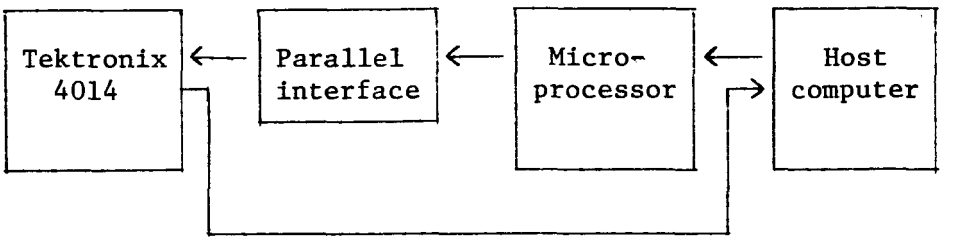


FIGURE 2

Any machine capable of producing GRAF80 commands may be the host computer. This could be the same or another microprocessor, a minicomputer, or a large mainframe. In our case a PRIME-300 minicomputer is used.

SOFTWARE

Software for GRAF80 consists of two main components, one on each computer. The microprocessor component receives graphic commands, stores them in a segmented data structure, and interprets them to produce either a refreshed or a stored image on the 4014. The host computer software consists of a package of Fortran subroutines which issue the commands.

MICROPROCESSOR SOFTWARE

Software for the microprocessor is written in PL/M, a high level language designed for microprocessors [2]. The software is designed to intercept and interpret a set of primitive graphical commands. It requires 4000 bytes of microprocessor memory plus buffers for the segmented display data structure generated from the commands. These commands consist of move and draw graphic output primitives, along with commands to OPEN, CLOSE, DISPLAY, and ERASE individual image segments. The complete list is given in Appendix B. They were selected from the SIGGRAPH-ACM CORE graphics package.

The main program consists of a loop which continuously monitors an I/O subtask and a graphics subtask. (The term "task" is used to indicate that these two processes could be executed in parallel if properly synchronized, although they have been implemented sequentially here. The net effect would be similar in either case.) When the main loop determines that a particular task has work to be done, it transfers control to that task, which runs to completion and returns to the main loop.

Input to the microprocessor is interrupt driven. A simple interrupt routine receives a character and stores it in a circular buffer. The job of the I/O task is to empty the buffer and evaluate its contents. In order to allow ordinary (non-graphical or graphical) use of the terminal, a "transparent" mode is defined. Transparent mode defaults until a special graphics notify character (ASCII ETB) is received. The I/O task examines each character and sends it directly to the terminal until an ETB is encountered, at which time further data is considered to be graphical. The graphical input is filtered out according to a predefined

transmission format (Appendix A) and stored as a picture segment. When the input buffer is emptied, control returns to the main loop.

The graphics task serves as an interpreter, converting the commands stored by the I/O task into device-dependent instructions which are sent to the 4014 to generate the picture. By redefining this task, different devices may be handled within the framework of a single system. Each call to this device driver interprets a single picture segment; repeated calls with the same segment allow the refresh capability. The graphics task returns to the main loop after once interpreting each segment which needs to be displayed.

Picture segmentation is handled at a level above the device driver. A number of graphics commands are used for manipulating segments. The I/O task intercepts these and takes appropriate action, such as setting flags, allocating storage, etc. The actual picture commands (or "primitives") are stored in the microprocessor's free memory as one or more 64-byte blocks, linked together by a forward chain, one chain per segment. A set of memory management routines is available to manipulate this data structure. Memory allocation is automatic and dynamic, implying that each segment receives only as much memory as it requires. The total amount of information which may be displayed simultaneously is thus limited by the amount of memory available to the microprocessor. Although the current implementation uses only real memory, this need not be the case; non-refreshed segments could use an auxiliary disk, but system performance would be somewhat degraded. Currently up to 16 segments may be defined.

Segment commands are defined below:

OPEN(N,R)--opens segment N. All subsequent graphic primitives become part of the segment, until a matching CLOSE is encountered.
R-refresh status -- 0 - no refresh
-- 1 - refresh

CLOSE(N) -- close segment N. No further information may be added to this segment.

DISPLAY(N)-cause segment N to be displayed.

ERASE(N) -- segment N is removed from the display, but may be re-displayed later.

DELETE(N)--segment is deleted from microprocessor memory.

Open segments may not be nested or overlapping; no more than one segment may be open at any time. No portion of a segment is visible until an explicit DISPLAY command is given, and a currently open segment cannot be displayed. This differs from the SIGGRAPH-ACM CORE system which specifies that portions of the open segment become immediately visible as they are defined. Due to the existence of both stored and refreshed segments in our implementation, we departed from the SIGGRAPH CORE here in order to considerably simplify the microprocessor software.

Erasing a segment merely removes it from the display; it remains available in the microprocessor memory for further use. A DELETE command destroys the segment entirely.

Refreshing, as pointed out, is accomplished by the graphics task; successive iterations of the main program loop refresh the segment. Since the I/O task usually is not busy, the graphics task becomes the limiting factor in providing flicker-free refresh. Within that task, two considerations limit the refresh capacity:

(a) the rate at which graphic commands can be interpreted into Tektronix code.

(b) the drawing speed of the Tektronix 4014 (which is rather slow for refresh purposes).

I/O time is negligible because of the parallel interface. Currently the flicker-free refresh capacity is approximately 75 characters or 65 inches of 2-inch vectors (including both dark and written vectors). This is about half the capacity of the Tektronix 4014 and is limited in our current implementation by the microprocessor software. Different software, discussed in the last section, should remedy this situation.

Non-refresh segments differ in that they are drawn only once, and not on each successive loop iteration. Normally such segments will be drawn in storage mode, but this does not have to be true. The primitive attribute REFRESH determines the mode which the Tektronix 4014 uses when generating characters and vectors; therefore it is possible to draw non-refreshed information which immediately disappears, and to draw storage-mode information which is continually refreshed (not recommended). The general assumption, however, is that refreshed segments will contain refresh commands and non-refresh segments will contain storage commands.

Selective erase is implemented somewhat differently for each type of segment. Refreshed segments merely have their display flag turned off. Non-refresh (storage) segments which are erased cause the entire display to be erased, and the display flag for that segment is turned off; all other segments are flagged as not having been drawn yet. The result is that everything but the erased segment is redrawn on the next pass of the main loop. The redraw time is very short, the only objection being the screen erase flash which precedes it.

HOST SOFTWARE

A package of FORTRAN interface subroutines on our PRIME-300 minicomputer allows a user's application program to generate commands for the GRAF80 system. Access is allowed to each segment-level and primitive-level

command. The subroutines generate the proper transmission formats, perform data conversions where necessary, and send GRAF80 commands out the communications line to the microprocessor. All coordinates used at this level are integers, which run from 0-4095. This corresponds directly to Tektronix 4014 enhanced graphics screen coordinates, although any arbitrary units could have been chosen. Appendix C contains documentation of these routines.

CONCLUSIONS

From the host computer's viewpoint the system appears as a virtual device, with built-in functions for picture segmentation and refreshing. As such, it would be most useful when interfaced to a higher-level graphics system. This higher-level system could be an existing graphics package, modified to accept GRAF80 as another device type and to include segment manipulation features; or it could be a new system built on top of GRAF80. In the latter case an implementation of the SIGGRAPH-ACM CORE system would be facilitated since GRAF80 has several of the CORE functions already available.

Our 8080 microprocessor is only lightly loaded with the current GRAF80 software (except when refreshing). Thus we believe that much of the CORE system could be handled by the microprocessor with additional software. Currently we have designed but not yet implemented on our microprocessor the image transformation features of the CORE system and most of the input functions. We chose these functions for microprocessor implementation since they can be most useful when placed on a processor with high data rate to the display device in order to provide fast interactive feedback to the user. Also, they involve only fixed point coordinates. Today's microprocessors handle floating point computations only

slowly in software. We have left the transformations from floating point (World coordinates) to fixed point (Normalized device coordinates) to our minicomputer which has floating point instructions.

As mentioned above, the limiting refreshing rate for GRAF80 could be improved through revision of the data structures involved. The graphics task could be modified to store Tektronix 4014 code in a separate buffer, rather than display it directly. A third refresh task would then display this buffer, with no interpretive overhead. A small test program on our microprocessor which used this technique was able to refresh approximately 140 inches of vectors, about twice as much as GRAF80 can currently refresh. However, our experience so far has been that the segmenting features (which provide selective erase) are more valuable for most users than the refresh capability.

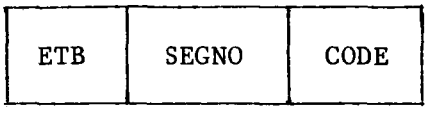
The handful of users with engineering applications who have been exposed to GRAF80 have generally been quite pleased with its capabilities. This is in part due to their previous use of an overloaded mainframe (a not uncommon occurrence) with a slow (300 to 2400 baud) data rate line to their terminal. These conditions favor our use of local intelligence and sending only changes in images instead of sending the entire image each time a small change is made.

APPENDIX A

TRANSMISSION FORMAT FOR GRAF80 COMMANDS

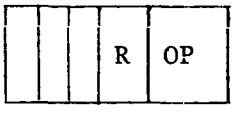
1. All characters received are passed transparently until the special control character ETB is encountered. (ETB = decimal 23)

2. Segment-level commands have the following 3-byte format:



SEGNO is the segment number, 0-15.

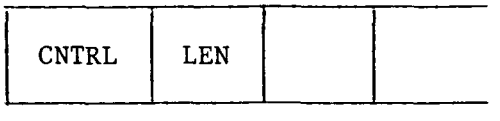
CODE is further broken down:



OP =
- 0 - OPEN *
- 1 - CLOSE
- 2 - DISPLAY
- 3 - ERASE
- 4 - DELETE

* The high order four bits of code are applicable only with the OPEN command. A "1" in the R bit signifies a refresh segment.

3. After an OPEN, all commands have the format:



. . . (LEN bytes)

CNTRL <> 23 - primitive graphic commands follow.
CNTRL = 23 - return to segment-level processing, (2) above.
0 < = LEN < = 255.

APPENDIX B

GRAF80 PRIMITIVE COMMANDS

All (X,Y) coordinates are assumed to be binary fractions in the range 0 to 1, with 12 bits of precision. The coordinate format is

X - LOW 8	X - HIGH 4	Y - LOW 8	Y - HIGH 4

using 4 bytes total. The most significant 4 bits of the HIGH bytes should be zeros. The various primitives are listed below, with the following format:

MNEMONIC OPERAND(S) BINARY OPCODE

REMARKS

MOVE X,Y 00010001

Move beam to (X,Y).

LINE X,Y 00010010

Draw line from current position to (X,Y), using current linestyle, refresh, and boldness.

POLYLINE N,X1,Y1,X2,Y2,...,XN,YN 00010011

Draw N connected lines, starting at the current position and ending at XN,YN. Use current linestyle, boldness, and refresh.

MARKER N,X,Y 00010100

Place a marker at (X,Y). If N=0, the marker is a point; otherwise, N is taken to be an ASCII character, and is drawn using the smallest charsize and the current refresh status.

TEXT N,STRING 00011000

Place text STRING of N characters at the current position. Use the current refresh status, boldness, and charsize.

SET ATTRIBUTE TYPE 00100000

TYPES are:

BOLDNESS	0000000x	x=1,bold **
		x=0,normal
LINESTYLE	000001xx	xx=00,solid
		xx=01,dotted
		xx=10,dot-dashed
		xx=11,dashed
CHARSIZE	0001hhww	hh=height
		ww=width

ww is ignored. Character sizes are based on hh, with 00 the largest and 11 the smallest.

REFRESH	1000011x	x=0,no refresh **
		x=1,refresh

** On the Tektronix, REFRESH and BOLDNESS are not mutually exclusive; REFRESH changes BOLDNESS, and BOLDNESS changes REFRESH.

STOP 00001111

Signifies the end of a stream of graphic primitives, and causes a return from the graphics task.

APPENDIX C

PRIME SUBROUTINES

The following routines generate segment-level commands as previously described.

```
CALL OPEN(N,IR)
CALL CLOSE(N)
CALL DISP(N)
CALL ERASE(N)
CALL DELETE(N)
```

Routines listed below generate primitive commands.

CALL MOVE(IX,IY)

Generates a MOVE command to coordinate position (IX,IY).

CALL DRAW(IX,IY)

Generates a LINE from the current position to (IX,IY).

CALL POLY(IXA,IYA,N)

Produces a POLYLINE primitive. IXA and IYA are arrays containing N coordinate values. N must be less than 64.

CALL MARKER(IX,IY,N)

Draws a marker at (IX,IY). N is either 0, denoting a point, or an ASCII character code.

CALL TEXT(ITXT,N)

Generate a text string at the current position. ITXT is an array containing N characters, packed two to a word.

CALL ATTRIB(IATTR,N)

Generates a SET ATTRIBUTE primitive. IATTR is the attribute to be set, and N is its value.

```
IATTR=1  -boldness      N=0   -normal
                         =1   -bold
```

IATTR=2	-linestyle	N=0	-solid
		=1	-dotted
		=2	-dot-dashed
		=3	-dashed
IATTR=3	-charsize	N=0	-largest
		=3	-smallest
IATTR=4	-refresh	N=0	-refresh off
		=1	-refresh on

CALL QUIT

Generates a STOP command. The CLOSE routine automatically generates a STOP code, so QUIT is not normally needed.

In addition, several routines are available which provide higher level features.

CALL LINE(IX1,IY1,IX2,IY2)

Draws a line from (IX1,IY1) to (IX2,IY2).

CALL RECTAN(IX,IY,IXSIZE,IYSIZE)

Draws a rectangle with its lower left corner at (IX,IY). IXSIZE and IYSIZE are the length and height, respectively, in GRAF80 coordinate units.

CALL NUM(I,IA,N)

Converts the integer value I to an ASCII character string which is returned in the array IA. N is the number of characters stored in IA. Characters are packed two to a word. IA may be used in a subsequent call to TEXT to display the string.

References

[1] Computer Graphics (11,3) Fall 1977.

[2] A Guide to PL/M Programming, Intel Corp., Santa Clara, California, 1973.